Final Report

September 1976

NATIONAL SOFTWARE WORKS DEVELOPMENT

By: BEVERLY BOLI

Prepared for:

ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE, NEW YORK

ARPA ORDER NO. 2853

STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 · U.S.A.

Final Report *September 1976*

NATIONAL SOFTWARE WORKS DEVELOPMENT

By: BEVERLY BOLI

Prepared for:

ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE, NEW YORK

ARPA ORDER NO. 2853

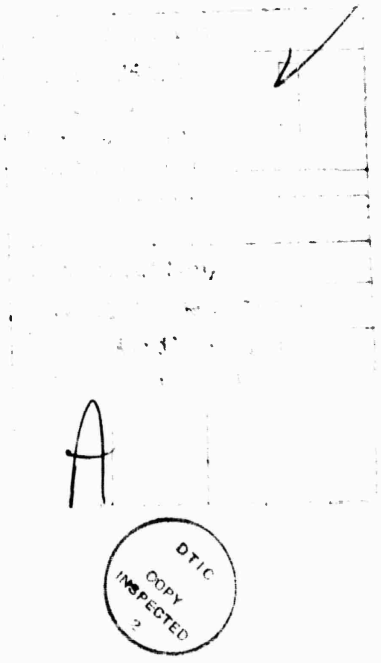
CONTRACT NO. F30602-75-C-0320

SRI Project 4051

Approved by:

D. C. ENGELBART, *Director*
Augmentation Research Center

EARLE D. JONES, *Executive Director*
Information Science and Engineering Division



Approved for public release;
distribution unlimited.

NATIONAL SOFTWARE WORKS DEVELOPMENT

FINAL PROJECT MANAGEMENT REPORT

Contractor: Stanford Research Institute

Contract Number: F30602-75-C-0156

Effective Date of Contract: 18 July 1974

Expiration Date of Contract and Amendments: 17 July 1975

Amount of Contract: $701,635

Program Code Number: FQ7619

SRI Project Number: 4051

Principal Investigator: Richard W. Watson
Phone: (415) 326-6200, ext. 2013

Project Engineer: Duane L. Stone
Phone: (315) 330-3857

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by D. L. Stone, RADC (ISIM), GAFB, NY 13440 under Contract F30602-75-C-0156.

CONTENTS

PREFACE 1

I The National Software Works Frontend System 3

II Protocols 13

III NLS 17

IV NLS Documentation for the NSW 22

REFERENCES 24

PREFACE

The goal of the NSW Program is to create a framework for sharing computer resources based on a communication network and constructed to support tools that aid every phase of software development[1]. Four basic components make up the NSW system: 1a

(1) The NSW Frontend system that provides terminal access to the ARPANET, a set of services creating a coherent NSW user environment, and an environment to decrease the cost of new tool creation. 1a1

(2) The Works Manager that provides special services such as authentication, record keeping, file system and file transfer, and management aids. 1a2

(3) The protocols and conventions needed between the Frontend and Works Manager, Frontend and tools, and Works Manager and tools. 1a3

(4) The tools (computers and software) that reside in the NSW. 1a4

The intention of the NSW system is to provide the user access to a number of general or specialized tools, so that the command discipline he uses remains constant even though the particular vocabulary changes from tool to tool as appropriate to describe that tool's functions. This user interface will not only reside on a PDP-10 but will also be available on a dedicated Frontend computer (PDP-11) for better responsiveness and less expense. We anticipate that heavily used tools or commands will, in time, actually be executed in the Frontend computer. In addition to increased system responsiveness, this will reduce network communication and will afford users some insulation from network or large computer unavailability. 1b

The tasks undertaken by the Augmentation Research Center for this contract period were of two types: those that contributed to the building of the NSW framework and those that provided initial tools--and improvements on those tools--for the NSW environment. The first two chapters of this report, "The National Software Works Frontend System' and "Protocols", describe progress made in the first task area, the NSW framework; the last two chapters, "NLS" and "NLS Documentation", discuss accomplishments in the tool task area. Each chapter provides a brief outline of the work completed in this contract

period. More detailed discussions may be found by consulting the "References". lc

I THE NATIONAL SOFTWARE WORKS FRONTEND SYSTEM

Introduction 2a

The Frontend provides the logical or conceptual function of interfacing a user to the NSW (Works Manager and tools). The initial Frontend consists of a PDP-11 satellite, providing terminal control and command language services, and a Help system running on a remote PDP-10. The Frontend also runs on a PDP-10. Frontend and Works Manager services are a logical whole as seen by the user and constitute an NSW Executive. The Frontend system is to provide the following types of services to the NSW: 2a1

(1) A formal language, the Command Meta Language (CML), for specifying NSW user interfaces. 2a1a

(2) A compiler for that formal language that runs under TENEX as a subsystem or from NLS and produces tool grammar data structures. 2a1b

(3) Tool grammars, products of the CML compiler or any other such program. 2a1c

(4) A command language interpreter (CLI) that processes a tool grammar in order to work with the user in specifying syntactically correct commands to the NSW. 2a1d

(5) A user profile data base that is used by the CLI while interacting with the user. This data base allows the Frontend to be tailored to the individual preferences of the users. 2a1e

(6) A user statistics data base. This Frontend element is currently not implemented, primarily because the statistics set has not been selected yet. The purpose is to accumulate statistics on user command usage and error rates. 2a1f

(7) Access to a semantic Help tool which is employed by the Frontend when the user requests semantic level help with a tool or a command. It is presumed that each tool, in addition to supplying the Frontend with a grammar, will also supply it with the name of a Help data base whose structure and content, as with the grammar, are the sole responsibility of the tool builder/supplier. 2a1g

This chapter provides an overview of each of the following Frontend elements: Command Meta Language (CML), Command Language Interpreter (CLI), compactor, communication mechanisms, and the semantic Help and Userprofile facilities. Progress made under this contract is discussed in the "FE-10 and FE-11 Status" section. More detailed explanations of specific accomplishments in the Frontend are available in documentation listed in References 1 through 14. Particularly useful is Irby [2]. 2a2

The Command Meta Language 2b

The Command Meta Language (CML) is a language for describing the command syntax of the user interface to application programs. CML syntax relies on two simple concepts--alternation (denoted by /) and succession (indicated by juxtaposing elements) [3, 4]. For example, a command with branching choices in it may be represented by a command word, followed by several command words separated by a /. Further interactions are specified by following elements. CML semantics are composed of built-in functions, semantic conventions, and parse functions. 2b1

The CML produces the full syntactic description of a command. The Backend uses execution functions to perform the low-level semantics of the command. In other words, the CML describes how the command "looks" to the user, rather than what it does inside the tool. 2b2

The CML program defining the user interface for a tool is compiled by the CML compiler (written using ARC's tree-meta compiler system [3, 4]) to produce object code (called a grammar), which is interpreted by a Command Language Interpreter (CLI). The Command Language Interpreter is cognizant of the device dependent feedback and addressing characteristics of the user's terminal and it manipulates the terminal, according to the grammar, in a way most suitable for that device. The grammar author need not know the type of terminal that will be used. 2b3

Extensions made to the Command Meta Language during this contract period are discussed in detail in Irby [5]. 2b4

The Command Meta Language Machine - the Command Language Interpreter (CLI) 2c

The Frontend's command parsing capability could have been implemented in several ways; for example, table driven techniques or a subroutine per command approach could have been used. A pseudo-computer design was selected [2]. The tool grammar serves as the program and the Command Language Interpreter executes this set of instructions. 2c1

This approach has several advantages: it is general; the tool dependent part (grammar) is relatively small; and when found inadequate, the CLI instruction set can be expanded. The generality and flexibility of the CLI were especially important design considerations. 2c2

A grammar, derived from a CML program and the CML compiler, is actually a pseudo-computer program that embodies the command language discipline for a tool. The CLI embodies the interaction discipline for all grammars. Thus, while commands (and operands) may be quite different from one tool to another, the user always follows the same discipline in specifying, aborting, accepting, rejecting, interrupting, and so on. 2c3

The CLI interacts with the user to help him specify commands for the system to execute. It prompts him for the type of input required (if the user wants it to), shows him the syntactic form of specific commands on request, indicates his actual alternatives at any point in the specification of a command on request, and can invoke a semantic help facility if the user requests. 2c4

The CLI provides a terminal-independent interface to tools. Because the CLI handles all terminal interaction, the tool need only address itself to that interface. The CLI will present to the tool one of a small number of virtual terminal classes. Thus, once a tool is developed, little attention need be given to the type or particular characteristics of the terminal employed while using the tool. The callable Frontend procedures are described in Irby [6]. 2c5

This means that even though the creators of a tool envisioned the user sitting at a typewriter terminal, the user who happens to be using a display terminal with a pointing device may be able to interact with the tool in a two-dimensional

sense, pointing to arguments on his screen instead of typing them. For tools that make more extensive use of a display terminal, the CML interpreter presents primitives for allocating windows on the display and allows the tool to manipulate displayed items and items off the scr... . In addition, the creators of a display-oriented tool can program in contingencies to handle a typewriter terminal, and even present different commands to the typewriter user where necessary. 2c6

Semantic Help Facility 2d

The Frontend helps the user by providing command word recognition, noise words, prompts, a "next option" list, and grammar syntax. All of these are based on the grammar and contain no semantics beyond the meaning of the command names and noise words. 2d1

If the above facilities are not sufficient because of uncertainty about a basic concept or vocabulary word, or the user wishes more information about the effects or use of a command, he can type the HELP key to enter the Help tool [7]. The Help facility is implemented as a special tool rather than a Frontend function. When the Help tool is invoked, command state information is passed to the Help tool, so it can take the user to an initial point that describes the command and field where he is located. 2d2

Once in the Help tool, a simple set of command conventions and the organization of the data base allow the user to easily reference related subjects, move to new subjects, or move up to higher-level descriptions. 2d2a

The highly structured Help data base is derived from the grammar syntax and text provided by the tool implementers that attempts to describe in English the intended use of the various commands and the tool as a whole. 2d3

Userprofile Facility 2e

The Userprofile facility consists of two parts, a data structure and a tool [8]. Together they give the user the ability to tailor the Frontend-user interaction to suit his proficiency and/or preferences. 2e1

The first part, the profile, is a data structure loaded by the Frontend when the user is authenticated by the Works Manager. It may be unique to each individual user, describing to the CML interpreter how much prompting and feedback the user wants, what recognition scheme he wishes to use to select command word alternatives, and many other idiosyncratic features of the user interface. The second part is the Userprofile Tool, which allows the user to modify his profile data base and consequently the behavior of the system. 2e2

Pseudo Telnet 2f

Pseudo Telnet is a program that supports Telnet-type interactions with "old tools" running in the NSW. (We use the term old tool to refer to a tool built before or outside of the NSW environment that cannot fully utilize all NSW facilities.) This program enables communication between the user and the tool as if the NSW did not exist. 2f1

The Command Language Interpreter Running on Different Machines 2g

The CLI is written in L10, a high level system programming language [9, 10]. L10 was originally designed for the DEC PDP-10, but the language is quite general. To implement the CLI on the PDP-11, we wrote an L10 compiler for the PDP-11 (L10l1) [11, 12]. 2g1

We considered means other than writing another L10 compiler. A complete discussion of the factors considered is provided in Andrews [13]. 2g2

CLI functions were divided into operating system independent and dependent parts. The independent parts exist in one set of sources, compiled by L10 for the PDP-10 and by L10l1 for the PDP-11. The dependent parts consist of special code and system calls in a special program for each computer. We have found this to be a very satisfactory and practical approach. 2g3

The Compactor 2h

The compactor accepts as input a compiled CML grammar and outputs the grammar in a much shortened form. This is one of several measures taken to make the most efficient use of space in the Frontend on both the PDP-10 and PDP-11. In addition to reducing the size of a grammar, the compactor separates the grammar into a large section which may be shared among many

users and a smaller section which must be private to each user. The ability to share grammars is particularly important on the PDP-11 to maximize the number of users the system can support. One additional function of the compactor is to put the result in the most suitable form for a given target machine (PDP-10 or PDP-11). 2h1

The compactor has been shown to provide a size reduction of one-half to one-third, depending on the nature of the grammar. The ability to share most of a grammar greatly reduces the total space needed when several users access the same grammar. 2h2

FE-10 and FE-11 Status 2i

Command Language Interpreter--An initial version of the NSW Command Language Interpreter (CLI) was designed and written for the PDP-10 and PDP-11 for typewriter-like terminals. The first release of the CLI for the PDP-10 was made on May 15, 1975. The CLI is now able to parse all commands for the works Manager, the NLS Editor tool, and some commands for other tools including the NSW debugger. The CLI communicates with tool processes and the Works Manager via the Distributed Programming System (DPS) or via a shared page. 2i1

Associated with the CLI is a formal language, called the Command Meta Language (CML), and its compiler. The output of this compiler serves as the program that is interpreted by the CLI in the course of interacting with the user. 2i1a

L10 Compiler for the PDP-11--Since it was required that the CLI be able to execute on either a PDP-11 or a PDP-10, a cross-compiler L10l1 was written to execute on a PDP-10 and compile L10 language programs into PDP-11 instructions. The CLI is written in L10, to be compiled and run on either a PDP-10 or a PDP-11. 2i2

In the course of writing the L10l1 compiler, it was necessary and desirable to incorporate into the formal L10 language new constructs that provided for machine-independent data structure declaration and manipulation as well as better exception handling mechanisms. Signals were improved and catchphrases were added to enhance the exception handling capabilities. Coroutines were added in a simple but general way, to allow procedures to communicate in this mode as well as normal call and returns. List data types and manipulation

constructs were also added, primarily to handle interhost data exchanges. 2i2a

Because of the differences in the PDP-10 and PDP-11, it was necessary to add elements to the L10 language to allow a particular L10 source program to function effectively on both a PDP-10 and a PDP-11 [9, 11, 12]. For example, the construct ADDRESS is an 18-bit quantity on the PDP-10 and a 16-bit quantity on the PDP-11. 2i2b

The enhanced version of the L10 compiler has been quite thoroughly debugged and the L10l1 compiler is currently being optimized to produce fewer instructions of code for the same high level constructs. In addition, the runtime package has been designed and implemented. 2i2c

Operating System Interface--Since the environment is quite different for the two machines, we also developed a level of software called the Operating System Interface (OSI). An OSI has been written and debugged for the PDP-10 and has been partially written for the PDP-11. The CLI has been compiled for the PDP-11 and debugging is now in progress. 2i3

ELF Operating System--SRI was the first major user of ELF for support of its own work. It was necessary for ARC staff to invest some time in shaking down the ELF operating system for use in the NSW. ELF now serves as a reliable terminal support system for ARC and as a development vehicle for CLI development. 2i4

NSW Debugger--The NSW debugger has been designed and partially coded and debugged [14, 15]. To facilitate NLS and CLI development activities, a special debugging package was provided that performs all its functions within a structure very similar to that of the eventual debugger and thus serves as a testbed for some of the basic low level primitives and approaches. 2i5

Stand-Alone PDP-11 Debugging Environment 2i6

To facilitate initial L10l1 testing and CLI debugging, a stand-alone debugging and cross-net loading facility was developed. This is now being modified to use under Elf. 2i6a

Future Plans 2j

Many extensions and refinements to the CLI are possible. Several reasonable follow-on projects are presented here. 2j1

A macro facility would allow a user to define and invoke macro-commands. Once defined, the user could invoke a macro anywhere in the command stream. Advanced versions would allow parameters, which would look like tool commands but would be expanded by the CLI in such a way that grammars and tools would not know a macro was being used. It is important that macros be easily and quickly invoked without, for example, running a special tool to invoke them. 2j2

The macro concept creates a storage problem for the PDP-11 Frontend system. Very small macros could be stored in local memory, but to be really useful the system should involve disk storage, or possibly some other form of readily accessible storage. 2j2a

A "meta tool" facility would allow the user to perform tasks that were actually sequences of operations involving several tools. The "meta tool" would look like a single tool to the user. In fact, it would consist of a grammar that called upon several tools in succession, or perhaps at once in some cases. The grammar could contain very high-level and specialized commands. For example, the user might edit, compile, execute and view results with one command. The meta-commands might invoke restricted sets of commands from tools the user did not normally have access to. This feature would not necessarily involve changes to the tool Backends. 2j3

Another useful feature might be tool interaction. This would be accomplished by a CLI mechanism(s) that allowed tool interaction that was definitely not projected by the tool implementers. This would allow the user, for example, to run two tools with a split screen and to specify source entities from one tool (e.g., TECO lines) as input to a command in the other tool (e.g., NLS statement). 2j4

A terminal-linking feature is an important addition. It should be possible to link display terminals as well as a display and a typewriter terminal in one of several ways: watch, duplex link, collaborate, and advise. 2j5

A more advanced Help facility could be devised to present information to the user, based on command error/usage statistics. It might be possible to make recommendations to the user about changing his profile, on the basis of those statistics. Interface to the BBN Scholar services should also be explored. 2j6

It is clear that the Frontend can play a key role in recovery after a system crash affecting one or more NSW components. This is a complicated problem. Here are a few examples of possible recovery actions: 2j7

(1) If a Tool Bearing Host (TBH) running a user's tool crashes, but his Frontend is still intact, it can start that tool again, perhaps on another host. It may not be possible to get the user to exactly the same state as at the time of the crash; the extent to which that can be done depends on the tool. It will usually be unreasonable to try to reproduce everything the user did up to that point. 2j7a

(2) If a TBH running the Works Manager (WM) crashes, the Frontend may be able to continue without the WM for a while, or it may have to reissue a call on the WM. In either case, the user's state should not be affected. 2j7b

(3) If the Frontend computer crashes there are several problems. The newly started Frontend must find out whether there were interrupted NSW users, and the logical place for that information is the Works Manager data base. If there were such users, it cannot be assumed that a user is still waiting faithfully at the same terminal. The user must ask to be reinstated as an active NSW user. The Frontend should find out from the WM what tool(s) he was using, load the appropriate grammars, and establish communication with the appropriate tool processes. The user will most probably be in a state as if he had aborted a command. 2j7c

(4) Most of these capabilities depend on appropriate information being accessible from the WM, such as user information and active tool information. It would also be helpful for the tool to give the WM enough information so that the tool could reconstruct its state at a later time. 2j7d

An obviously desirable goal is to make the CLI faster and more efficient and enable it to service more users on a PDP-11

Frontend. However, it is difficult to say what can be done here until studies of a running system are made. 2j8

II PROTOCOLS

Introduction 3a

The Distributed Programming System (DPS), as designed for use in the National Software Works, is an environment for programs that utilize multiple computers. A program may call on functions or procedures in several computers as easily as it calls on procedures in one computer. A more detailed explanation of the concepts and motivation for the Distributed Programming System is given in White [16]. 3a1

Past Protocol Design Approaches 3b

Several applications protocols have been designed and implemented since the Host-Host Protocol was adopted. Most have been bootstrapped from lower-level protocols. For example, the File Transfer Protocol (FTP) was built upon Telnet, and the Remote Job Entry Protocol (RJE) upon both Telnet and FTP. The highest-level protocol shared by all such bootstrapped protocols is Telnet. Although the bootstrapping principle seems a sound basis for Network protocol development, we believe that Telnet, providing little more of use than a character set, is not the most appropriate foundation for many applications protocols. 3b1

Bootstrapping at a Higher Level 3c

We contend that a Procedure Call Protocol (PCP)--a Network-standard mechanism for invoking arbitrarily named, argument-driven, and result-producing procedures in a remote process--is an appropriate and powerful foundation for many applications protocols. We believe that the adoption by the Network community of a PCP as the basis for most applications protocols would have at least the following effects: 3c1

(1) Expedite the specification of applications protocols by permitting their documentation to have a functional, rather than a syntactic orientation. 3c1a

(2) Largely eliminate the need for separate, application-specific user processes. 3c1b

(3) Reduce the cost of making large, existing software systems available as Network servers by allowing a Network interface more compatible with their internal organization. 3c1c

(4) Provide the basis for a more natural interface between local and remote procedures. 3c1d

(5) Encourage, therefore, the sharing of software, by making procedures on remote hosts as accessible to the programmer as local ones. 3c1e

Procedure Call Protocol and Multiprocess Systems 3d

The Procedure Call Protocol (PCP) is an interprocess and/or interhost protocol that permits a collection of processes within one or more ARPANET hosts to communicate at the procedure call level. In effect, it makes the component procedures of remote software systems as accessible to the programmer as those witnin his own system. PCP defines the distributed programming system and the interprocess exchanges that implement it. 3d1

At the highest level, PCP is the specification of a virtual programming environment in which remote procedures are assumed to operate. The model specifies the manner in which remote procedures gain and relinquish control, the kinds of data structures with which they can be expected to deal, and so forth. One of the tasks of the PCP implementer, therefore, is to provide a mapping between his real programming environment and the virtual one defined by PCP. 3d1a

At a slightly lower level, PCP is the specification of the interchanges between two connected processes that implement the virtual programming environment. 3d1b

PCP is intended to be suitable for interlinking two processes within a single host, as well as processes on different hosts. 3d1c

A multiprocess system whose construction PCP makes practical, and of which the NSW is an example, consists of collections of "procedures" and "data stores" called "packages", in "processes", interconnected in a tree structure by "channels". Procedures witnin a process have free access to the procedures (and data stores) of each process adjacent to it in the tree structure, and may call upon them as if they were local subroutines. Superimposed upon the tree structure is a more general set of interconnections that give nonadjacent processes in the tree the same kind of access to one another. 3d2

A multiprocess system is implemented by: 3d3

(1) Low-level protocols which provide the basic, interprocess communication (IPC) facilities and data structure format specifications. 3d3a

(2) A PCP proper, which defines the procedure call and return mechanism and specifies the interprocess control exchanges required. 3d3b

(3) A set of system procedures which, implemented within each process, provides mechanisms by which user procedures can call remote procedures, manipulate remote data stores, and interconnect processes. 3d3c

(4) User packages in each process. 3d3d

The Procedure Call Protocol can be viewed as a set of system primitives provided by the collection of computer operating systems cooperating in the multiprocess system. 3d4

Application Packages 3e

Besides a mechanism for calling procedures, a programmer needs the capability to group related procedures together and to institute the execution of groups of procedures as processes. Such groupings of related procedures are called packages. The package is the unit of information that we allow to be access protected. We expect that normally there will be one package to a process, though this is not a requirement. Any process may consist of several packages, and the selection of packages may change dynamically. We expect that packages will be defined and implemented to group functionally related capabilities; for example, we have specified a file package. 3e1

We envision that the procedures developed for one application will be used in others. Eventually there could be PCP callable libraries of procedures. In fact, we believe that such "procedure packages" as UCLA's BMDs, or IBM's Scientific Subroutine Package should be PCP-callable procedures. 3e2

Accomplishments 3f

We have designed, specified, and implemented a PCP and the beginning of a multiprocess system called the Distributed Programming System (DPS). This implementation is documented

for applications programmers in White [17], and the specification of the interface to other DPS implementations is given in White [18]. The implementation is in the programming language L10. An index to the program procedures is provided in White [19]. The source code itself is in White [20]. 3f1

We have designed and specified several application packages: an Executive Package [21], a Batch Job Package [22], a Low-Level Debug Package [23], and a File Package [24]. This File Package received extensive attention and revision as the NSW developed [25]. We also designed and documented a model for remote job entry in NSW [25]. 3f2

Most of the documents describing the version 2.0 of the PCP and application packages and protocols are collected in two volumes: Postel, Watson, and White [27] and Postel and White [28]. In addition there are three general papers that discuss the Distributed Programming System [16, 29, 30]. 3f3

III NLS

Introduction 4a

Enhancement of NLS as a flexible and efficient tool integrated into the NSW environment was one of the four major tasks undertaken by the Augumentation Research Center. The conversion from NLS-7 to NLS-8 improved the user interface and accomplished the first step toward the Frontend/Backend split (NLS-9) that would make NLS a fully integrated NSW tool. NLS-9 design was completed and NLS-8 grammar converted to work with the new CML/CLI. In addition, new features were added to NLS and a test version, NLS-8.5, was brought up. A more complete discussion of these accomplishments follows. 4a1

NLS-7 to NLS-8 Conversion 4b

NLS went through a major design change during the previous contract period. The goals of the NLS-8 design were to make the user interface simpler, more flexible, consistent throughout the system, and easier to use. In addition, the use of the Command Meta Language (CML) for the user interface module allowed experiments with different command language structures and feedback and simplified building subsystems. Finally the design anticipated moving the "frontend" functions to a minicomputer. 4b1

This conversion of NLS-7 to NLS-8 was completed. It involved coding the user interface portions of NLS in the new CML. NLS-8 was brought up as the running system first at ARC, then, in November 1974, at Office-1. All NLS subsystems (the Ident system, Sendmail, Journal delivery, the Calculator, and Programs), as well as all Class 1 user programs, were also converted. 4b2

Several documents were written as aids to users to familiarize them with the new system [31, 32], and special training sessions were given to make the transition easier. 4b3

An on-line Help system was implemented and the Help data base was completed [7, 33]. The syntax command was written to allow the user to display the syntax of any NLS command. This command was extended to produce a file containing the syntax of all commands which can then be used to update the Help data base or to produce a listing, such as the "NLS-8 Command Summary" [34]. 4b4

NLS-9 4c

NLS-9 was designed to take advantage of the full set of capabilities presented by the NSW environment: Frontend, Protocols, and Works Manager. This was done to provide a test bed for NSW concepts for tools specifically developed or modified for the NSW environment and to provide the economies and responsiveness offered by the NSW Frontend. 4c1

A study of the Encapsulator capabilities and restrictions indicated that installing NLS-9 under the Encapsulator would give unsatisfactory performance [25]. 4c2

The design for splitting NLS code into separate Frontend and Backend components was completed. An extensive and detailed document was written describing all Backend routines callable by the Frontend [35]. This document was intended as a reference work for anyone wishing to interface another NSW tool directly to the NLS Backend as well as a working document for converting the NLS code. (References to ISI are in the document because ISI was considering using the NLS Backend code.) 4c3

The NLS-9 grammar was converted to work with the new CML/CLI. The Backend routines for the editor and programs systems were rewritten as needed to compile with the new L10ll compiler [11, 12]. (A more detailed list may be found in Martin [36].) A first version of NLS-9 was constructed with the grammar (Frontend) and Backend running in separate forks and communicating by means of a shared page. This version allowed us to begin debugging the new grammar and Backend routines. 4c4

An interface module (the "middle end") was written to allow Frontend/Backend communication through DPS [37, 38]. Version 2 of NLS-9 was brought up using DPS, and this version was then used for continued debugging. This version was consistent with the Procedure call protocol (PCP) provided by DPS and allowed (during the follow-on contract period) transition to MSG, the replacement protocol for DPS, without changing the actual NLS backend routines, although the interface module had to be rewritten. 4c5

Design of the NLS interaction with the Works Manager file system was performed as completely as possible (given available Works Manager documentation), and coding was started. However, no debugging was possible because the Works

Manager file primitives were not yet available. The definition of the WM primitives continued in a state of flux [39]. 4c6

New Features and NLS-8.5 4d

The NSW contract specified a number of new features for the version of NLS (NLS-9) to be installed in the NSW. We decided to develop these enhancements in a new version of NLS-8, NLS 8.5, for two reasons. First, we needed a stable environment to test and debug. Second, we wanted final testing to be done by actual users working on real projects. New features were added to NLS-8.5 in the areas described below: 4d1

COBOL--In ARC's original statement of work major emphasis was given to adding features to NLS that would directly benefit COBOL programming [40, 41]. Subsequent conversations with Col. McGovern, Data Services Center personnel, and NSW steering committee members indicated that document production tools were of much greater initial interest. 4d2

Consequently, a minimal system was designed that allows a programmer to write a COBOL program in NLS, process the NLS file performing certain edits and extensions, and, finally, to create a Tenex sequential file in proper COBOL format. This file is suitable for submission to the Works Manager's Remote job entry tool. The major effort was directed toward document production tools. 4d2a

NLS File System--NLS's structured file system was generalized to include two new concepts, the property list and the inferior tree [42]. Properties are typed data blocks that are chained to the nodes within an NLS file. The familiar NLS statement has a property of type text. In addition to the text property, a property was added to allow the storage of line drawings within a file. This concept can be extended to include such properties as editorial comments, digitized speech strings, and heading entries. 4d3

NLS Graphics--The capability for the creation and manipulation of line drawings was added to NLS. The lineprocessor was modified to service a standard storage tube graphic display. Using the mouse and keyset, the graphics user can create and edit drawings, which may themselves contain text, and store the drawings within an NLS file. A library of standard flow

chart symbols was made to aid programmers. A draft of the Graphics commands is provided [43]. 4d4

A virtual COM representation for drawings was designed and the output processor was modified to create COM files containing mixed text and graphics. 4d5

Second COM Device--with George Lithograph Co., software was developed for the Singer 6000 COM machine to process virtual COM files. Character tables were incorporated into the output processor to accommodate the Singer fonts. 4d6

COM Proofs--A user subsystem was written to allow a virtual COM file to be displayed on the TEKTRONIX 4014. The file can then be proofread for desired page layout before sending the file to a COM vendor. 4d7

Multifile Help System--The NLS Help system was redesigned and coded to work across multiple file data bases, and to serve as the NSW Help system for any tool that provides the necessary data base. New tool data bases were added (e.g. Calculator), and revisions on the existing data base were started. 4d8

File Conversion Software--Software was written to convert IBM EBCDIC files to ASCII Tenex files on a PDP-10 and to convert the Tenex files to structured NLS files. This allowed us to begin a pilot project on a rewrite of a very large Air Force manual (AFM 66-1) that had been entered on MTST tape cartridges. 4d9

Formatting Systems--A subsystem was written to automatically format NLS files according to Air Force specifications (provided by the Data Services Center). The system inserts Output Processor directives in the file to produce a properly formatted document both on the line printer and for COM. 4d10

Tabs--A second mode of handling tabs was implemented. The new mode replaces tabs typed by the user with the appropriate number of spaces in the file. It allows a user to specify right, left, or centered justification of text to a tab stop. 4d11

Table Manipulation--A user system was written to augment the manipulation of tabular material. The system allows the user to do such things as transpose columns and easily adjust the spacing between columns. 4d12

Automatic Editing--Working with the Pentagon Data Services Center personnel, we identified a number of commonly made errors in entering text. We then wrote a subsystem to find and correct these errors. They include such things as incorrect number of spaces following punctuation marks, incorrect number of spaces at the beginning of sentences, and changing the case of certain words. 4d13

Table of Contents--The Air Force formatting system was extended to generate a table of contents for a multifile document in the specified Air Force format. 4d14

Pilot Project--A very large Air Force manual, AFM 66-1, was chosen by the Data Services Center personnel to be used as a test case for using NLS for document production. The text of the 12 volumes was entered on MTST cartridges, read into Tenex sequential files, and converted to structured NLS files. As the new features listed above became ready for testing they were used by SRI staff members to produce the first three volumes of the manual. This use in a production environment suggested modifications and extensions, which were made. 4d15

IV NLS DOCUMENTATION FOR THE NSW

Online and hardcopy documentation of NLS are both produced for the NSW. Online documentation takes the form of the Help data bases. Hardcopy ranges from a short pamphlet "TNLS-8 Quick Reference" to the loose-leaf bound "Secretarial Functions Guide".

5a

The NLS-8 Help data base was divided into a multifile data base to better organize the growing amount of information about NLS [44]. The Help directory now contains the "NLS" file, covering concepts and terminology common to all NLS subsystems and programs, and separate files for each subsystem. A "Publications" file was also created to act as an index for several subsystem files for use in publications work. New subsystem data bases were added (e.g., Calculator data base) and revisions on the existing NLS data base and individual subsystem data bases were begun.

5b

The NLS-8 Glossary was produced in hardcopy form from the Help data base [45]. This is a complete source of information pertaining to the use of NLS. Terms peculiar to NLS, or more general computer terminology that may be unfamiliar to new users without computer backgrounds, are listed in alphabetical order with explanations and extensive cross-referencing.

5c

The Secretarial Functions Guide, a self-instructional aid to performing secretarial functions using NLS tools, was completed [46]. The Guide was printed in hardcopy form and bound in a loose-leaf notebook to allow each user to include only the documents applicable to his needs, and to allow future additions and updates to be incorporated easily. It introduces the user to NLS and explains how to perform such tasks as writing and sending memos, creating and revising drafts of reports, and writing and formatting letters. Most of the documents are called Sample Sessions, which contain a brief introduction, an instructional section that takes the user step by step through the specific commands necessary to perform a task, and a summary of all of the commands taught in that session.

5d

Other documents include:

5e

Preface to NLS Tools--An introduction to basic NLS concepts and commands [46].

5e1

NLS-8 Command Summary--A complete listing of all NLS-8 commands and their syntax, classified by subsystem [34]. 5e2

TNLS-8 Quick Reference--A very brief summary of the most frequently used NLS commands, conventions, and special characters for use at the terminal [47]. 5e3

Introduction to NLS Documentation--A discursive introduction outlining NLS capabilities for document production [45]. 5e4

Format Library--A guide to the various formats available through the Format subsystem [48]. 5e5

REFERENCES

National Software Works

1. Robert Balzer, T E Cheatham, Stephen Crocker, Stephen Warshall. The National Software Works. Information Sciences Institute, Marina del Rey, Ca. 20-DEC-75. (24716,)

Frontend

CML

2. Charles H Irby. The Command Meta Language System. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 6-Jan-76. (27266,)

3. Augmentation Research Center. Tree Meta Report--Preliminary Draft. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 23-JAN-73. (14045,)

4. Augmentation Research Center. Tree Meta Report--Preliminary Report, Formal Description. Stanford Research Institute, Menlo Park, Ca. 23-JAN-73. (14046,)

5. Charles H Irby. CML Extensions for the NSW. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 10-JAN-75. (25056,)

Frontend Procedures

6. Charles H Irby. NSW Packages and Procedures. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 17-JUL-75. (26151,)

Help

7. Harvey G Lehtman, Kirk Kelley, Dirk H van Nouhuys, Jeanne M Beck. Query/Help Software and Data Bases. Knowledge Workshop Development -- Reported as of 7/74. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 31-DEC-75. (22133,)

Userprofile

8. Raphael Rom. Userprofile Users' Guide. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 23-JAN-76. (27404,)

L10

9. Augmentation Research Center. L10 Users' Guide. Stanford Research Institute, Menlo Park, Ca. 8-JAN-75. (34211,)

10. William H Paxton. L10 Documentation. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 29-MAY-71. (7052,)

11. Don I Andrews. Extensions to the L10 Programming Language for the DEC PDP-10 and DEC PDP-11. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 4-FEB-75. (25295,)

12. Don I Andrews. L10 List Documentation. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 4-APR-75. (25692,)

13. Don I Andrews. Factors in L10/BCPL Language Decision for NSW Frontend. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 14-NOV-74. (24519,)

Debugger

14. Ken E Victor. The Design and Implementation of a Multi-Process, Multi-Machine, Multi-Language Interactive Debugger. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. (27487,)

15. Ken E Victor. Programmers' Guide to the Debugger. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 22-SEPT-75. (26529,)

Distributed Programming System

16. James E White. A High-Level Framework for Network-Based Resource Sharing. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 23-DEC-75. (27197,)

17. James E White. DPS-10 Version 2.5 Programmer's Guide. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 13-AUG-75. (26271,)

18. James E White. DPS Version 2.5 Implementer's Guide. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 15-AUG-75. (26282,)

19. James E White. DPS-10 Version 2.5 Procedure Directory (SYSGD). Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 15-AUG-75. (26283,)

20. James E White. DPS-10 Version 2.5 Source Code. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 13-AUG-75. (26267,)

21. Jonathan B Postel. EXEC 2 / The Executive Package. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 22-NOV-74. (24580,)

22. Jonathan B Postel. BATCH 2 / The Batch Job Package. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 22-NOV-74. (24583,)

23. James E White. LLDBUG 2 / The Low-Level Debug Package. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 22-NOV-74. (24579,)

24. Jonathan B Postel. NSW Files -- Package, Format, Types, Movement, Conversion. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 9-MAY-75. (25850,)

25. Jonathan B Postel. Dispatcher and Encapsulator. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 6-JUN-75. (25978,)

26. Jonathan B Postel. The NSW Remote Job Entry Model (DRAFT). Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 4-DEC-74. (24655,)

27. Jonathan B Postel, Richard W Watson, and James E White. Procedure Call Protocol Documents. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 30-DEC-74. (24855,)

28. Jonathan B Postel and James E White. National Software Works PCP Documents. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 30-DEC-74. (24856,)

29. Jonathan B Postel and James E White. Notes on a Distributed Programming System. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 21-MAR-75. (25613,)

30. James E White. Elements of a Distributed Programming System. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 6-JAN-76. (27250,)

NLS

System Documentation

31. Jeanne M Beck. Details of NLS-7 to NLS-8 Conversion. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 22-SEP-74. (23386,)

32. Dirk van Nouhuys. Brief Document to Aid Transition from Old to New TNLS. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 28-AUG-74. (23886,)

33. Michael D Kudlick. Help Data Base Design. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 13-SEPT-73. (19062,)

34. Augmentation Research Center. NLS-8 Command Summary. Stanford Research Institute, Menlo Park, Ca. 16-MAY-75. (24831,)

35. Karolyn J Martin. Definition of PCP Callable Routine in the NLS Editor--Primarily for ISI. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 5-FEB-75. (25307,)

36. Karolyn J Martin. Summary of NLS Backend Changes for NLS-9. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 14-JAN-76. (27343,)

37. David S Maynard. The Middle End: A Protocol Interface. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 17-OCT-75. (26694,)

38. Susan K Ocken. NLS-9 Argument Conversions. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 26-JAN-76. (27411,) (Note--this is an update of: David S Maynard. NLS-9 Arguments Conversion. (26744,))

39. Larry L Garlick. Questions About Works Manager Capabilities. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 10-DEC-75. (27128,)

40. Larry A Crain. COBOL Programming in NSW. AFDSDC, Gunter AFS, Montgomery, Ala. 13-MAY-75. (32066,)

41. Elizabeth K Michael, et. al. NSW/NLS Plans. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 21-NOV-74. (24570,)

42. Harvey G Lehtman. Developments in the NLS File System for NSW. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 9-JAN-76. (27292,)

43. Robert L Belleville and Susan K Ocken. Draft of Graphics Command Summary. Augmentation Research Center. Stanford Research Institute, Menlo Park, Ca. 28-JAN-76. (27438,)

User Documentation

44. Augmentation Research Center. Help Data Bases. Stanford Research Institute, Menlo Park, Ca. On-line user documentation for NLS, available at ISIC and OFFICE-1. The following files, containing command descriptions and explanations on how to use each NLS subsystem, are located in the Help directory and may best be viewed using the Help command:

NLS (Index file)
AFMFormat
Base
Calculator
Decimal
Format
Graphics
Modify
Programs

Proof
Publication
Publish
Sendmail
Useroptions

45. Augmentation Research Center. NLS-8 Glossary. Stanford Research Institute, Menlo Park, Ca. 21-NOV-75. (22132,)

46. Augmentation Research Center. Secretarial Functions Guide. Stanford Research Institute, Menlo Park, Ca. 10-SEP-75. Comprised of the following documents:
Title Page and Introduction. <26442,>
Preface to NLS Tools. <26320,>
Editing Sample Session I. <23911,>
Editing Sample Session II. <26422,>
Editing Sample Session III. <26423,>
File-Viewing Sample Session. <26424,>
Help Services Sample Session. <26425,>
Sendmail Sample Session I. <26426,>
Sendmail Sample Session II. <26427,>
Format Sample Session. <25925,>
Application Tutorial for Letter Production and Multiple Copy Production and Distribution. <33404,>
Introduction to Documentation through NLS. <26429,>

47. Augmentation Research Center. TNLS-8 Quick Reference. Stanford Research Institute, Menlo Park, Ca. 21-APR-75. (25765,)

48. Augmentation Research Center. Format Library. Stanford Research Institute., Menlo Park, Ca. 6-MAR-75. (25764,)